OIPE
FEB 25 2005
JC88
PATENT & TRADEMARK OFFICE

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicants: Baerlocher, Anthony J., et al.
Appl. No.: 09/919,022
Conf. No.: 3718
Filed: July 31, 2001
Title: GAMING DEVICE WITH BONUS SCHEME HAVING MULTIPLE SYMBOL MOVEMENT AND ASSOCIATED AWARDS
Art Unit: 3714
Examiner: Reassigned to Michael W. O'Neill
Docket No.: 112300-820

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

AFFIDAVIT OF ANTHONY J. BAERLOCHER UNDER 37 C.F.R. § 1.131

Sir:

I, Anthony J. Baerlocher, being duly sworn, hereby state:

1. I am employed by IGT as a Director of Game Design.

2. I am one of the named inventors of the subject matter claimed in the above-identified patent application.

3. I am generally familiar with the subject matter claimed in the above-identified patent application, and I am generally familiar with the inventions disclosed and claimed therein.

4. I read the current claims of the above-referenced application. I believe that the claimed invention was conceived at least as early as December 23, 1998.

5. A partially redacted printout of the programming code for one commercial embodiment of the present invention is attached hereto as Exhibit A. This section of the programming code included in Exhibit A outlines the logic flow of one commercial

BEST AVAILABLE COPY

embodiment of the above-referenced patent application. The programming code is adapted to be executed by a processor which controls a gaming device and a display device. The processor of the gaming device executes the code to display the bonus game of the presently claimed invention.

More specifically, Exhibit A includes two different types of information: descriptive purpose sections and programming code sections. Before each section of programming code, the purpose section describes what the following section of programming code seeks to accomplish. The programming code is adapted to be executed by a processor to cause a display device to display a player symbol, represented by a fox symbol, which moves to at least one location on a path. For example the portion on pg. 7 of Exhibit A which begins with "Purpose: Spin fox reel, hilite the target tile, and move fox to target position" describes the code which follows which causes the fox symbol to move to a location. The processor executes the programming code to cause a display device to display a terminating symbol, represented by a hound symbol, to move to at least location on the path. For example, the purpose section on pg. 10 of Exhibit A beginning with "Purpose: spin hound reel and move hound to target tile" describes the code which follows which causes the hound symbol to move to a location. Exhibit A includes a plurality of award outcomes throughout the programming code.

6. The printout of the portion of programming code attached hereto as Exhibit A is a true and accurate document associated with the development of one commercial embodiment of the above-referenced patent application.

7. With respect to Claim 1, the attached code of Exhibit A, which was conceived at least as early as December 23, 1998 and was developed at least as early as December 15, 1999, is for a commercial implementation for a processor controlled gaming device which includes a display device wherein the gaming device is operable to: (a) cause a player symbol to visit at least one of a plurality of locations on a path; (b) display the player symbol visiting the location; (c) cause a terminating symbol to visit at least one of the locations on the path; (d) display the terminating symbol visiting the location; and (e) provide a player with any bonus value associated with the location visited by the player symbol.

8. With respect to Claim 23, the attached code of Exhibit A, which was conceived at least as early as December 23, 1998 and was developed at least as early as December 15, 1999, is for a commercial implementation for a processor controlled gaming device which includes a display device wherein the gaming device is operable to: (a) cause a player symbol to visit at least one of the locations on a path; (b) display the player symbol visiting the location; (c) cause a terminating symbol to visit at least one of the locations on the path; (d) display the terminating symbol visiting the location; and (e) provide a player with any outcome associated with the location visited by the player symbol.

9. With respect to Claim 26, the attached code of Exhibit A, which was conceived at least as early as December 23, 1998 and was developed at least as early as December 15, 1999, is for a commercial implementation for a method of gaming which includes the steps of: (a) triggering a bonus round; (b) displaying a path including a plurality of locations; (c) causing at least one player symbol to visit one of said plurality

of locations; (d) causing at least one terminating symbol to visit one of said plurality of locations; (e) awarding a player any bonus value associated with a location visited by a player symbol; and (f) repeating steps (c) through (e) until the player symbol catches the terminating symbol or the terminating symbol catches the player symbol.

10. With respect to Claim 37, the attached code of Exhibit A, which was conceived at least as early as December 23, 1998 and was developed at least as early as December 15, 1999, is for a commercial implementation for a method of gaming which includes the steps of triggering a bonus round, displaying a plurality of locations, associating a terminating symbol with at least one of the locations, providing at least one potential award, moving said terminating symbol at least once during the bonus round, and terminating the bonus round following a predetermined event.

11. With respect to Claim 41, the attached code of Exhibit A, which was conceived at least as early as December 23, 1998 and was developed at least as early as December 15, 1999, is for a commercial implementation for a method of gaming which includes the steps of triggering a bonus round, displaying a plurality of symbols, including at least one terminating symbol, moving the terminating symbol at least once during the bonus round, providing at least one potential award and terminating the bonus round following a predetermined event.

12. With respect to Claim 46, the attached code of Exhibit A, which was conceived at least as early as December 23, 1998 and was developed at least as early as December 15, 1999, is for a commercial implementation for a processor controlled gaming device includes a path including a plurality of locations, a first movable symbol displayed on one of the locations, a second movable symbol displayed on one of the

locations, predetermined location changes associated with the first movable symbol and the second movable symbol, a bonus value associated with at least one of the location changes, a termination event associated with the first movable symbol and the second movable symbol being positioned at an identical location a processor and a display device, electronically connected to the processor, which displays the path, the first movable symbol, and the second movable symbol to the player.

13. With respect to Claim 47, the attached code of Exhibit A, which was conceived at least as early as December 23, 1998 and was developed at least as early as December 15, 1999, is for a commercial implementation for a processor controlled gaming device which includes a memory device wherein the gaming device is operable to: (a) initiate a bonus round, (b) change the location of the first movable symbol, (c) display the changed location of the first movable symbol, (d) change the location of the second movable symbol, (e) display the changed location of the second movable symbol, (f) provide a player with a bonus value after the movable symbol is positioned at least one predetermined location, and (g) terminate the bonus round after the first movable symbol and the second movable symbol are positioned at an identical location.

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true, and further, I acknowledge that willful false statements and the like are punishable by fine or imprisonment, or both, under §1001 of Title 18 of the United States Code and may jeopardize the validity of the application or any patent issuing thereon.

Signature

Name: Anthony J. Baerlocher

Date Signed

18 Feb 2005

Address:

State of Nevada)
) SS.
County of Nevada)

SUBSCRIBED and SWORN before me this 18 day of February, 2004.

Notary Public

KARYN TRABERT
Notary Public - State of Nevada
Appointment Recorded in Washoe County
No: 97-3234-2 - Expires August 14, 2005

Exhibit A
Redacted

```
/* (c) Copyright 1998 International Game Technology */

/*
     $Workfile:  foxs.cpp  $
     $Archive:  V:/arch/mslaveii/type28/bgm00095/foxs.cpv  $
     $Revision:  1.8  $
     $Date:  22 Nov 1999 15:31:16  $
     $Author:  ptruong  $

   Purpose:
     Game states for Fox 'N Hound (a Type28 protocol game)

   Modifications:
     Author        Date      Explanation
     ------------- --------- ------------
     Phong Truong   26 April 99  Created
*/


#include "foxs.hpp"
#include "foxg.hpp"
#include "foxact.hpp"
#include "igtdefs.hpp"
#include "utils.hpp"
#include "m_effect.hpp"


#define GAME ((FoxNHoundGame *) game)   // Defined to simplify code

////////////////////////////////////////////////////////////////////////////////
// FoxNHoundStateBonusIdle                                                    //
//                                                                            //
////////////////////////////////////////////////////////////////////////////////
//----------------------------------------------------------------------------
[redacted]
{
}

//----------------------------------------------------------------------------
[redacted]
```

Exhibit A
Redacted

[redacted]

[redacted]

[redacted]

```
// Purpose: Present the shg reel animation
void FoxNHoundStateBonusIdle::onIdle (void)
{
   switch (getSubstate ())
      {
      case SUBSTATE_INIT:
         setSubstate (SUBSTATE_ANIMATE_DONE);
                        break;

                case SUBSTATE_ANIMATE_DONE:
         DEBUGLOG ("FoxNHoundStateBonusIdle::onIdle()\n");
         GAME->ackMessage(PACKET_BONUS_WIN_ACK, GAME-
>BGS_STATE_IDLE);
         setSubstate (SUBSTATE_IDLE);
         break;

      case SUBSTATE_IDLE:
      default:
          break;
      }
```

Exhibit A
Redacted

```
}


////////////////////////////////////////////////////////////////////////////////
// FoxNHoundStateBonusIntroduction                                            //
//                                                                            //
////////////////////////////////////////////////////////////////////////////////
//-----------------------------------------------------------------------------
// Purpose: Construct this class.
FoxNHoundStateBonusIntroduction::FoxNHoundStateBonusIntroduction
(FoxNHoundGame *game)
   : Type28StateBonusIntro (game)
{
        setSubstate (SUBSTATE_IDLE);
}

//-----------------------------------------------------------------------------
// Purpose: Synchronize this state.
[redacted]

//-----------------------------------------------------------------------------
[redacted]

//-----------------------------------------------------------------------------
[redacted]


//-----------------------------------------------------------------------------
// Purpose: First bonus message.  Fox shows excitement
void FoxNHoundStateBonusIntroduction::onIdle (void)
{
   switch (getSubstate ())
      {
```

Exhibit A
Redacted

```
      case SUBSTATE_INIT:
         GAME->gmcloud->hide ();
         GAME->gmtext[GAME->gmtext_index]->hide ();
         GAME->show_gmtext = FALSE;
         GAME->number_gm_show_text = 0;
         setSubstate(SUBSTATE_ANIMATE_DONE);
         break;

      case SUBSTATE_ANIMATE_DONE:
         GAME->foxStationaryHideFlic ();
         GAME->foxhop->hide ();
         DEBUGLOG("FoxNHoundStateBonusIntroduction::onIdle()\n");
```

[redacted]

[redacted]

```
////////////////////////////////////////////////////////////////////////////////
// FoxNHoundStateBonusWait                                                    //
//                                                                            //
////////////////////////////////////////////////////////////////////////////////
//------------------------------------------------------------------------------
// Purpose: Construct this class.
```

[redacted]

```
//------------------------------------------------------------------------------
// Purpose: Synchronize this state.
void FoxNHoundStateBonusWait::synchronizePresentation (void)
{
   Type28StateBonusWaitForInput::synchronizePresentation ();

   if (GAME->bPowerIsUp)
      GAME->bPowerIsUp = FALSE;

   GAME->meter_array[WIN_METER]->updateGraphicMeter (0);
   GAME->meter_array[WIN_METER]->show ();
```

Exhibit A
Redacted

```
  GAME->meter_array[LAP_METER]->updateGraphicMeter (1);
  GAME->meter_array[LAP_METER]->show ();
  setSubstate (SUBSTATE_INIT);
}

//-----------------------------------------------------------------------------
void FoxNHoundStateBonusWait::enter (void)
{
  Type28StateBonusWaitForInput::enter();
}

//-----------------------------------------------------------------------------
void FoxNHoundStateBonusWait::exit (void)
{
  GAME->waitforspin->hide ();
  GAME->stopIntroSound ();

  if (!GAME->foxexcite->isStopped())
    GAME->foxExciteHide ();

  Type28StateBonusWaitForInput::exit();
}

//-----------------------------------------------------------------------------
// Purpose: Second bonus message.  Popup wait-for-spin
void FoxNHoundStateBonusWait::onIdle (void)
{
  switch (getSubstate ())
  {
    case SUBSTATE_INIT:
      GAME->smallHndStationaryHide ();
      GAME->startIntroSound();
      setSubstate (SUBSTATE_ANIMATE_FOX_GOCIRCLE);
                  break;

    case SUBSTATE_ANIMATE_FOX_GOCIRCLE:
      if (GAME->foxGoCircle ())
        setSubstate(SUBSTATE_ANIMATE_FOXHOUND_POSITION);
      break;

    case SUBSTATE_ANIMATE_FOXHOUND_POSITION:
      GAME->getFoxPositionData (GAME->POSRESET);
      GAME->getHoundPositionData (GAME->POSRESET);
      setSubstate(SUBSTATE_ANIMATE_POSITION_DONE);
      break;
```

```
      case SUBSTATE_ANIMATE_POSITION_DONE:
         if (GAME->isTimeToShowFNH())
            setSubstate(SUBSTATE_ANIMATE_INREADY_POSITION);
         break;

      case SUBSTATE_ANIMATE_INREADY_POSITION:
         GAME->waitForSpinReel ();
         setSubstate(SUBSTATE_ANIMATE_INREADY_POSITION_DONE);
         break;

      case SUBSTATE_ANIMATE_INREADY_POSITION_DONE:
         setSubstate(SUBSTATE_ANIMATE_DONE);
         break;

      case SUBSTATE_ANIMATE_DONE:
         DEBUGLOG ("FoxNHoundStateBonusWait::onIdle()\n");
         GAME->ackMessage(PACKET_BONUS_WIN_ACK, GAME-
>BGS_STATE_WAIT_FOR_INPUT);
         setSubstate(SUBSTATE_IDLE);
         break;


      case SUBSTATE_IDLE:
      default:
          break;
   }
}



////////////////////////////////////////////////////////////////////////////////
// FoxNHoundStateBonusPlay                                                    //
// FOX play                                                                   //
////////////////////////////////////////////////////////////////////////////////
//-----------------------------------------------------------------------------
// Purpose: Construct this class.
[redacted]


//-----------------------------------------------------------------------------
```

Exhibit A
Redacted

```
// Purpose: Synchronize this state.
void FoxNHoundStateBonusPlay::synchronizePresentation (void)
{
   Type28StateBonusPlayA::synchronizePresentation ();

   GAME->foxhop->hide ();
   GAME->getHoundPositionData (GAME->POSPREVIOUS);
   GAME->getHoundPositionData (GAME->POSCURRENT);
   GAME->getFoxPositionData (GAME->POSPREVIOUS);
   GAME->getFoxPositionData (GAME->POSCURRENT);
   GAME->showWinAnimateHide ();

   GAME->hilitiles[GAME->fox_tile_pos]->hide ();
   GAME->foxStationaryShow (GAME->fox_tile_prev_pos, FALSE);
   GAME->smallHndStationaryShow (GAME->hound_tile_pos);
   GAME->showMeters();
   GAME->showLaps();

   if (GAME->bPowerIsUp)
      GAME->bPowerIsUp = FALSE;

   setSubstate (SUBSTATE_INIT);
}

//-----------------------------------------------------------------------------
```

[redacted]

```
//-----------------------------------------------------------------------------
void FoxNHoundStateBonusPlay::exit (void)
{
    Type28StateBonusPlayA::exit();
}

//-----------------------------------------------------------------------------
// Purpose: Spin fox reel, hilite the target tile, and move fox to target position
void FoxNHoundStateBonusPlay::onIdle (void)
{
   switch (getSubstate ())
   {
      case SUBSTATE_INIT:
         if (GAME->theme_sound->isStopped())
            GAME->startThemeSound();
```

Exhibit A
Redacted

```
        setSubstate(SUBSTATE_ANIMATE_FOXREEL_SPIN);
        break;

    case SUBSTATE_ANIMATE_FOXREEL_SPIN:
        GAME->spinFoxReel();
        setSubstate(SUBSTATE_ANIMATE_FOXREEL_SPIN_DONE);
        break;


    case SUBSTATE_ANIMATE_FOXREEL_SPIN_DONE:
        if (GAME->spinFoxReelDone())
        {
            GAME->foxhop->hide ();
            setSubstate(SUBSTATE_ANIMATE_FOXMOVE);
        }
        break;


    case SUBSTATE_ANIMATE_FOXMOVE:
        if (GAME->foxMoveNTiles ())
            setSubstate(SUBSTATE_ANIMATE_ENDGAME);
        break;


    case SUBSTATE_ANIMATE_ENDGAME:
        if ((GAME->fox_pos_prop >= GAME->PROPERTY_FOXCAUGHTHOUND)
&& (GAME->fox_pos_prop < GAME->PROPERTY_LAPS_BONUS))
        {
            if (GAME->lapsmeter >= MAX_LAP || GAME->fox_pos_prop >= GAME-
>PROPERTY_FOXCAUGHTHOUND_HOUNDATGO)
            {
                if (GAME->hound_tile_pos > 0)                    //if hound is not at go, do
cloud animation
                    setSubstate(SUBSTATE_ANIMATE_WIN_ENTER);      //
                else                                             //
                    setSubstate(SUBSTATE_ANIMATE_DONE);
            }
            else
                setSubstate(SUBSTATE_ANIMATE_WIN_ENTER);         //do cloud
animations
        }
        else
            setSubstate(SUBSTATE_ANIMATE_DONE);
        break;
```

Exhibit A
Redacted

```
    case SUBSTATE_ANIMATE_WIN_ENTER:
      GAME->reelsHide ();
      GAME->foxhop->hide ();
      GAME->foxStationaryHideFlic ();
      GAME->smallHndStationaryHide ();
      setSubstate(SUBSTATE_ANIMATE_WINMOVE);
      break;

    case SUBSTATE_ANIMATE_WINMOVE:
      if (GAME->collideMoveNTiles ())
      {
            GAME->meter_array[WIN_METER]->show ();
        GAME->startTime = system->getTimerMilliseconds ();
        setSubstate(SUBSTATE_ANIMATE_WINMOVE_DELAY);
      }
      break;

    case SUBSTATE_ANIMATE_WINMOVE_DELAY:
      if (GAME->isCollideMoveNTilesDelayDone ())
        setSubstate(SUBSTATE_ANIMATE_WINMOVE_DONE);
      break;

    case SUBSTATE_ANIMATE_WINMOVE_DONE:
      GAME->atGoalLap1Play ();
      setSubstate(SUBSTATE_ANIMATE_ATGOAL);
      break;

    case SUBSTATE_ANIMATE_ATGOAL:
      if (GAME->atGoalLap1Stop ())
        setSubstate(SUBSTATE_ANIMATE_DONE);
      break;

    case SUBSTATE_ANIMATE_DONE:
      DEBUGLOG("FoxNHoundStateBonusPlay::onIdle()\n");
      GAME->ackMessage(PACKET_BONUS_WIN_ACK, GAME-
>BGS_STATE_PLAY_A);
      setSubstate(SUBSTATE_IDLE);
      break;

    case SUBSTATE_IDLE:
      break;

    default:
      break;
  }
}
```

Exhibit A
Redacted

```
////////////////////////////////////////////////////////////////////////////////
// FoxNHoundStateBonusPlayHound                                              //
//                                                                           //
////////////////////////////////////////////////////////////////////////////////
//-----------------------------------------------------------------------------
// Purpose: Construct this class.
[redacted]


//-----------------------------------------------------------------------------
// Purpose: Synchronize this state.
void FoxNHoundStateBonusPlayHound::synchronizePresentation (void)
{
  Type28StateBonusPlayB::synchronizePresentation ();

  GAME->getFoxPositionData (GAME->POSRESET);
  GAME->getHoundPositionData (GAME->POSPREVIOUS);
  GAME->getHoundPositionData (GAME->POSCURRENT);
  GAME->showWinAnimateHide ();
  GAME->showMeters();
  GAME->showLaps();

  if (GAME->houndland->isStopped())
  {
    GAME->houndmove->hide ();
    GAME->smallHndStationaryShow (GAME->hound_tile_prev_pos);
    GAME->foxStationaryShow (GAME->fox_tile_pos, FALSE);
  }

  if (GAME->bPowerIsUp)
    GAME->bPowerIsUp = FALSE;

  setSubstate (SUBSTATE_INIT);
}

//-----------------------------------------------------------------------------
void FoxNHoundStateBonusPlayHound::enter (void)
{
  Type28StateBonusPlayB::enter();
}
```

Exhibit A
Redacted

```
//-----------------------------------------------------------------------------
void FoxNHoundStateBonusPlayHound::exit (void)
{
   Type28StateBonusPlayB::exit();
}

//-----------------------------------------------------------------------------
// Purpose: spin hound reel and move hound to target tile
void FoxNHoundStateBonusPlayHound::onIdle (void)
{
  switch (getSubstate ())
     {
     case SUBSTATE_INIT:
       if (GAME->theme_sound->isStopped())
         GAME->startThemeSound ();

       setSubstate(SUBSTATE_ANIMATE_HOUNDREEL_SPIN);
       break;

     case SUBSTATE_ANIMATE_HOUNDREEL_SPIN:
       GAME->spinHoundReel ();
       setSubstate(SUBSTATE_ANIMATE_HOUNDREEL_SPIN_DONE);
       break;

     case SUBSTATE_ANIMATE_HOUNDREEL_SPIN_DONE:
       if (GAME->spinHoundReelDone ())
       {
         GAME->smallHndStationaryHide ();
         setSubstate(SUBSTATE_ANIMATE_HOUNDMOVE);
       }
       break;

     case SUBSTATE_ANIMATE_HOUNDMOVE:
       if (GAME->smallHndMoveNTiles ())
         setSubstate(SUBSTATE_ANIMATE_ENDGAME);
       break;

     case SUBSTATE_ANIMATE_ENDGAME:
       if (GAME->hnd_pos_prop == GAME->PROPERTY_HOUNDCAUGHTFOX)
       {
         GAME->reelsHide ();
         GAME->foxhop->hide ();
         setSubstate(SUBSTATE_ANIMATE_WIN_ENTER);
       }
       else
         setSubstate(SUBSTATE_ANIMATE_DONE);
```

```
        break;

    case SUBSTATE_ANIMATE_WIN_ENTER:
        GAME->foxStationaryHideFlic ();
        GAME->smallHndStationaryHide ();
        setSubstate(SUBSTATE_ANIMATE_WIN);
        break;

    case SUBSTATE_ANIMATE_WIN:
        GAME->houndmove->hide ();
        setSubstate(SUBSTATE_ANIMATE_WINMOVE);
        break;

    case SUBSTATE_ANIMATE_WINMOVE:
        if (GAME->collideMoveNTiles ())
        {
                GAME->meter_array[WIN_METER]->show ();
            GAME->startTime = system->getTimerMilliseconds ();
            setSubstate(SUBSTATE_ANIMATE_WINMOVE_DONE);
        }
        break;

    case SUBSTATE_ANIMATE_WINMOVE_DELAY:
        if (GAME->isCollideMoveNTilesDelayDone ())
            setSubstate(SUBSTATE_ANIMATE_WINMOVE_DONE);
        break;

    case SUBSTATE_ANIMATE_WINMOVE_DONE:
        GAME->atGoalLap1Play ();
        setSubstate(SUBSTATE_ANIMATE_ATGOAL);
        break;

    case SUBSTATE_ANIMATE_ATGOAL:
        if (GAME->atGoalLap1Stop ())
            setSubstate(SUBSTATE_ANIMATE_ATGOAL_DONE);
        break;

    case SUBSTATE_ANIMATE_ATGOAL_DONE:
        switch (GAME->theme_number)
        {
            case GAME->SHOW_ROUNDUP:
                GAME->roundupSeqPlay ();
                setSubstate(SUBSTATE_ANIMATE_ROUNDUP);
                break;

            case GAME->SHOW_JAILBOUND:
```

Exhibit A
Redacted

```
            GAME->jailboundSeqPlay ();
            setSubstate(SUBSTATE_ANIMATE_JAILBOUND);
            break;

          case GAME->SHOW_TOAST:
            GAME->toastSeqPlay ();
            setSubstate(SUBSTATE_ANIMATE_TOAST);
            break;

          case GAME->SHOW_DISGUISE:
          default:
            GAME->disguiseSeqPlay ();
            setSubstate(SUBSTATE_ANIMATE_DISGUISE);
            break;

        }
        break;

      case SUBSTATE_ANIMATE_DISGUISE:
        if (GAME->isDisguiseSeqContinue ())
          setSubstate(SUBSTATE_ANIMATE_DISGUISE_DONE);
        break;

      case SUBSTATE_ANIMATE_DISGUISE_DONE:
        if (GAME->isDisguiseSeqStop ())
          setSubstate(SUBSTATE_ANIMATE_DONE);
        break;

      case SUBSTATE_ANIMATE_ROUNDUP:
        if (GAME->isRoundupSeqContinue ())
          setSubstate(SUBSTATE_ANIMATE_ROUNDUP_DONE);
        break;

      case SUBSTATE_ANIMATE_ROUNDUP_DONE:
        if (GAME->isRoundupSeqStop ())
          setSubstate(SUBSTATE_ANIMATE_DONE);
        break;

      case SUBSTATE_ANIMATE_JAILBOUND:
        if (GAME->isJailboundSeqContinue ())
          setSubstate(SUBSTATE_ANIMATE_JAILBOUND_DONE);
        break;

      case SUBSTATE_ANIMATE_JAILBOUND_DONE:
        if (GAME->isJailboundSeqStop ())
          setSubstate(SUBSTATE_ANIMATE_DONE);
```

Exhibit A
Redacted

```
        break;

    case SUBSTATE_ANIMATE_TOAST:
        if (GAME->isToastSeq1Continue ())
            setSubstate(SUBSTATE_ANIMATE_TOASTSEQ);
        break;

    case SUBSTATE_ANIMATE_TOASTSEQ:
        if (GAME->isToastSeq2Continue ())
            setSubstate(SUBSTATE_ANIMATE_TOAST_DONE);
        break;

    case SUBSTATE_ANIMATE_TOAST_DONE:
        if (GAME->isToastSeqStop ())
            setSubstate(SUBSTATE_ANIMATE_DONE);
        break;

    case SUBSTATE_ANIMATE_DONE:
        DEBUGLOG("FoxNHoundStateBonusPlay::onIdle()\n");
        GAME->ackMessage(PACKET_BONUS_WIN_ACK, GAME-
>BGS_STATE_PLAY_B);
        setSubstate(SUBSTATE_IDLE);
        break;

    case SUBSTATE_IDLE:
    default:
        break;
  }
}

////////////////////////////////////////////////////////////////////////////////
// FoxNHoundStateBonusWin                                                     //
//                                                                            //
////////////////////////////////////////////////////////////////////////////////
//------------------------------------------------------------------------------
// Purpose: Construct this class.
[redacted]

//------------------------------------------------------------------------------
// Purpose: Synchronize this state.
[redacted]
```

Exhibit A
Redacted

[redacted]

```
//---------------------------------------------------------------------------------------
// Purpose: Get and show the fox total win meter upto target tile
void FoxNHoundStateBonusWin::onIdle (void)
{
   switch (getSubstate ())
   {
      case SUBSTATE_INIT:
         GAME->getFoxPositionData (GAME->POSCURRENT);
         GAME->getHoundPositionData (GAME->POSCURRENT);
         GAME->showMeters ();
         GAME->showMetersSound ();
         setSubstate(SUBSTATE_ANIMATE_DONE);

      case SUBSTATE_ANIMATE_DONE:
         DEBUGLOG("FoxNHoundStateBonusWin::onIdle()\n");
         GAME->ackMessage(PACKET_BONUS_WIN_ACK, GAME-
>BGS_STATE_WIN);
         setSubstate(SUBSTATE_IDLE);
         break;

      case SUBSTATE_IDLE:
      default:
          break;
   }
}

////////////////////////////////////////////////////////////////////////////////////
// FoxNHoundStateBonusShow                                                        //
//                                                                                //
////////////////////////////////////////////////////////////////////////////////////
//---------------------------------------------------------------------------------------
// Purpose: Construct this class.
```

[redacted]

Exhibit A
Redacted

[redacted]

```
//--------------------------------------------------------------------------------
// Purpose: Synchronize this state.
```

[redacted]

```
//--------------------------------------------------------------------------------
```

[redacted]

```
//--------------------------------------------------------------------------------
```

[redacted]

```
//--------------------------------------------------------------------------------
void FoxNHoundStateBonusShow::onIdle (void)
{
  switch (getSsubstate ())
  {
    case SUBSTATE_INIT:
      GAME->getFoxPositionData (GAME->POSCURRENT);
      GAME->showMeters ();
      GAME->showLaps();
      GAME->reelsHide ();
      GAME->foxhop->hide ();
      setSubstate(SUBSTATE_ANIMATE_SHOW);
      break;

    case SUBSTATE_ANIMATE_SHOW:
      GAME->showWinAnimateShow ();
      setSubstate(SUBSTATE_ANIMATE_DONE);
      break;

    case SUBSTATE_ANIMATE_DONE:
      DEBUGLOG("FoxNHoundStateBonusPlay::onIdle()\n");
```

Exhibit A
Redacted

```
        GAME->ackMessage(PACKET_BONUS_WIN_ACK, GAME-
>BGS_STATE_SHOW_WIN);
        setSubstate(SUBSTATE_IDLE);
        break;

    case SUBSTATE_IDLE:
    default:
        break;
  }
}
////////////////////////////////////////////////////////////////////////////////
// FoxNHoundStateBonusExtraBonus                                              //
//                                                                            //
////////////////////////////////////////////////////////////////////////////////
//-----------------------------------------------------------------------------
// Purpose: Construct this class.
[redacted]

//-----------------------------------------------------------------------------
// Purpose: Synchronize this state.
[redacted]

[redacted]

//-----------------------------------------------------------------------------
void FoxNHoundStateBonusExtraBonus::exit (void)
{
}

//-----------------------------------------------------------------------------
// Purpose: nothing to show here for this state
void FoxNHoundStateBonusExtraBonus::onIdle (void)
{
  switch (getSubstate ())
    {
```

Exhibit A
Redacted

```
    case SUBSTATE_INIT:
      GAME->smallHndStationaryHide ();
      GAME->reelsHide ();
            GAME->getFoxPositionData (GAME->POSCURRENT);
      GAME->showWinAnimateHide ();
      GAME->showMeters ();
      setSubstate(SUBSTATE_ANIMATE_WIN);
      break;

    case SUBSTATE_ANIMATE_WIN:
      if (GAME->lapsmeter >= MAX_LAP)
        setSubstate(SUBSTATE_ANIMATE_FOXWIN);
      else
      {
        if (GAME->fox_pos_prop == GAME->PROPERTY_FOXCAUGHTHOUND
|| GAME->fox_pos_prop == GAME-
>PROPERTY_FOXCAUGHTHOUND_HOUNDATGO)
        {
          GAME->stopCollideSound ();
          GAME->stopThemeSound();
          GAME->foxhop->hide ();
          GAME->houndmove->hide ();
          GAME->foxStationaryShow (0, FALSE);
          GAME->startTime = system->getTimerMilliseconds () + 2500;
          setSubstate(SUBSTATE_ANIMATE_FOXWIN_FLASHTILE);
        }
        else
          setSubstate(SUBSTATE_ANIMATE_THEME);
      }
      break;

    case SUBSTATE_ANIMATE_THEME:
      switch (GAME->theme_number)
      {
        case GAME->SHOW_ROUNDUP:
          GAME->roundupSeqPlay ();
          setSubstate(SUBSTATE_ANIMATE_ROUNDUP);
          break;

        case GAME->SHOW_JAILBOUND:
          GAME->jailboundSeqPlay ();
          setSubstate(SUBSTATE_ANIMATE_JAILBOUND);
          break;

        case GAME->SHOW_TOAST:
          GAME->toastSeqPlay ();
```

Exhibit A
Redacted

```
            setSubstate(SUBSTATE_ANIMATE_TOAST);
            break;

          case GAME->SHOW_DISGUISE:
          default:
            GAME->disguiseSeqPlay ();
            setSubstate(SUBSTATE_ANIMATE_DISGUISE);
            break;

        }
        break;

      case SUBSTATE_ANIMATE_DISGUISE:
        if (GAME->isDisguiseSeqContinue ())
          setSubstate(SUBSTATE_ANIMATE_DISGUISE_DONE);
        break;

      case SUBSTATE_ANIMATE_DISGUISE_DONE:
        if (GAME->isDisguiseSeqStop ())
          setSubstate(SUBSTATE_ANIMATE_DONE);
        break;

      case SUBSTATE_ANIMATE_ROUNDUP:
        if (GAME->isRoundupSeqContinue ())
          setSubstate(SUBSTATE_ANIMATE_ROUNDUP_DONE);
        break;

      case SUBSTATE_ANIMATE_ROUNDUP_DONE:
        if (GAME->isRoundupSeqStop ())
          setSubstate(SUBSTATE_ANIMATE_DONE);
        break;

      case SUBSTATE_ANIMATE_JAILBOUND:
        if (GAME->isJailboundSeqContinue ())
          setSubstate(SUBSTATE_ANIMATE_JAILBOUND_DONE);
        break;

      case SUBSTATE_ANIMATE_JAILBOUND_DONE:
        if (GAME->isJailboundSeqStop ())
          setSubstate(SUBSTATE_ANIMATE_DONE);
        break;

      case SUBSTATE_ANIMATE_TOAST:
        if (GAME->isToastSeq1Continue ())
          setSubstate(SUBSTATE_ANIMATE_TOASTSEQ);
        break;
```

```
    case SUBSTATE_ANIMATE_TOASTSEQ:
      if (GAME->isToastSeq2Continue ())
        setSubstate(SUBSTATE_ANIMATE_TOAST_DONE);
      break;

    case SUBSTATE_ANIMATE_TOAST_DONE:
      if (GAME->isToastSeqStop ())
        setSubstate(SUBSTATE_ANIMATE_DONE);
      break;

    case SUBSTATE_ANIMATE_FOXWIN:
      GAME->foxwinSeqFlashTilesPlay ();
      setSubstate(SUBSTATE_ANIMATE_FOXWIN_FLASHTILE);
      break;

    case SUBSTATE_ANIMATE_FOXWIN_FLASHTILE:
      if (GAME->isFoxwinSeqFlashTilesDone ())
      {
        GAME->foxwinWinHopPlay ();
        setSubstate(SUBSTATE_ANIMATE_FOXWIN_WINHOP);
      }
      break;

    case SUBSTATE_ANIMATE_FOXWIN_WINHOP:
      if (GAME->isFoxwinWinHopDone ())
        setSubstate(SUBSTATE_ANIMATE_FOXWIN_BIGWIN);
      break;

    case SUBSTATE_ANIMATE_FOXWIN_BIGWIN:
      if (GAME->isFoxwinBigWinDone ())
        setSubstate(SUBSTATE_ANIMATE_FOXWIN_FINALE);
      break;

    case SUBSTATE_ANIMATE_FOXWIN_FINALE:
      if (GAME->isFoxwinFinale ())
        setSubstate(SUBSTATE_ANIMATE_DONE);
      break;

    case SUBSTATE_ANIMATE_DONE:
      DEBUGLOG("FoxNHoundStateBonusExtraBonus::onIdle()\n");
      GAME->ackMessage(PACKET_BONUS_WIN_ACK, GAME-
>BGS_STATE_EXTRA);
      setSubstate(SUBSTATE_IDLE);
      break;
```

Exhibit A
Redacted

```
      case SUBSTATE_IDLE:
      default:
        break;
    }
}

/* (c) Copyright 1998 International Game Technology */

/*
      $Workfile:  foxg.cpp  $
      $Archive:  V:/arch/mslaveii/type28/bgm00095/foxg.cpv  $
      $Revision:  1.11  $
      $Date:  22 Nov 1999 15:31:10  $
      $Author:  ptruong  $

   Purpose:
      Game state machine and presentation layer for Fox 'N Hound (a Type28 protocol
game)

   Modifications:
      Author        Date      Explanation
      --------------- ---------- ------------
      Phong Truong    26 April 99  Created
*/

#include "foxs.hpp"
#include "foxact.hpp"
#include "foxg.hpp"
#include "gamepack.hpp"
#include "gswin.hpp"

//-----------------------------------------------------------------------------
static const int FOX_NHOUND_PAYTABLE_ID = 0;
static const int FOX_NHOUND_PAYTABLE_VALIDATION_ID = 0;
static const int CREDIT_ROLLUP_SOUND_DELAY = 3243 ;



//-----------------------------------------------------------------------------
// Purpose: Construct this class.
FoxNHoundGame::FoxNHoundGame (void):
   goW (0),                goY (0),              reelfram (0),          arrows (0),
   waitforspin (0),          foxreel (0),          foxhop (0),            foxexcite (0),
   sign5c (0),             flag100_top (0),      flag600 (0),           atgoal (0),
   foxreel_outcome (0),        houndreel(0),         houndmove (0),         houndland
(0),
```

houndreel_outcome (0), hilitiles (0), flag100_bot (0),
podium (0), flag500 (0), finishW (0), finishY (0),
foxsnk (0), truck (0), cloud (0), winhop (0),
bigwin (0), foxwin (0), foxtst (0), drnkin (0),
roundp (0), dogsut (0), lapsmeter (0), winamount (0),
theme_number (0), startTime (0), bFlashNextGrp (0), meter_array (0),
number_of_meters (0), stopTime (0), flashTime (0), coins_won(0),
system_time(0), betone_sounds (0), betmax_sounds (0), loss_sounds (0),
gmcloud (0), gmtext (0), gmtext_items (0), show_gmtext (FALSE),
gmtext_index (0), fox_tile_pos (0), fox_pos_prop(0), fox_tile_prev_pos (0),
fox_tile_reset_pos (0), foxreel_prev_pos (0), fox_gocircle_pos (0), foxreel_pos (0),
hound_tile_prev_pos (0), hound_tile_pos (0), hnd_pos_prop (0), houndreel_prev_pos (0),
houndreel_pos (0), collide_tile_pos (0), drnkin_loop (0), hound_tile_reset_pos (0),
foxexcite_sound (0), theme_sound (0), foxspin_sound (0), foxmove_sound (0),
houndspin_sound (0), rstop_sound (0), whine_sound (0), busted_sound (0),
roundup_sound (0), drinkin_sound (0), foxsneak_sound (0), bigwin_sound (0),
intro_sound (0), introlp_sound (0), houndclose_sound (0), foxaway_sound (0),
foxclose_sound (0), newAttenuation(0), tile100_sound (0), tile_sound (0),
tile0_sound (0), reel_sound (0), flash_sound (0), win5c_sound (0),
afterhoundspin_sound (0), houndmove_sound (0), collide_sound (0), gowin_sound(0),
hound_tooclose (FALSE),
fox_tooclose (FALSE), fox_getaway (FALSE), panic_flag (FALSE), fox_at_tile0 (FALSE),
fox_go_circle (FALSE), bPowerIsUp (FALSE), do_credit_at_tile0(FALSE), number_gm_show_text (0),
foxreel_outcome_items (0), hilitiles_items (0), do_delay_at_tile0(FALSE), houndreel_outcome_items (0),
do_foxexcite_at_tile0(FALSE), do_delay_credit(FALSE), play_win_5crless (FALSE),
loss_game_counter (0), number_of_betone_sounds (0),
number_of_betmax_sounds (0), number_of_loss_sounds (0),

Exhibit A
Redacted

```
   betone_sound_index (0),        credit_rollup_initiated(0),
   credit_rollup_sound_count(0),  credit_rollup_start_time(0),
   credit_rollup_sounds(0),       credit_rollup_tag_sounds(0),
   credit_rollup_time_delay(0),   credit_rollup_tag_sounds_index(0),
   number_of_credit_rollup_sounds(0),
   number_of_credit_rollup_tag_sounds(0),
   credit_rollup_substate(SUBSTATE_CREDIT_ROLLUP_IDLE)
{
}

//-----------------------------------------------------------------------------
// Purpose:
FoxNHoundGame::~FoxNHoundGame (void)
{

   delete [] foxreel_outcome;
   delete [] houndreel_outcome;
   delete [] hilitiles;
   delete [] gmtext;
   delete [] meter_array;
   delete [] credit_rollup_sounds;
   delete [] credit_rollup_tag_sounds;
   delete [] betone_sounds;
   delete [] betmax_sounds;
   delete [] loss_sounds;
}

//-----------------------------------------------------------------------------
```

```
//-----------------------------------------------------------------------------
```

```
//-----------------------------------------------------------------------------
```

Exhibit A
Redacted

```
// Purpose: Create the screen resources, cache commonly used actors.
void FoxNHoundGame::createPresentation (void)
{
   extern ActorConfigList fox_nhound_game_actors;

   Type28Game::createPresentation ();
   director->addActors (&fox_nhound_game_actors);
}

//-----------------------------------------------------------------------------
// Purpose: Create the screen resources, cache commonly used actors.
void FoxNHoundGame::initializePresentation (void)
{
   DWORD i;

   // set up background display for game idle state
   arrows     = (ActorVideo *) getActor (ACTOR_ARROWS);
   goW        = (ActorVideo *) getActor (ACTOR_GOW);
   goY        = (ActorVideo *) getActor (ACTOR_GOY);
   finishW    = (ActorVideo *) getActor (ACTOR_FINISHW);
   finishY    = (ActorVideo *) getActor (ACTOR_FINISHY);
   podium     = (ActorVideo *) getActor (ACTOR_PODIUM);
   flag100_top = (ActorVideo *) getActor (ACTOR_100FLAG_TOP_BMP);
   flag100_bot = (ActorVideo *) getActor (ACTOR_100FLAG_BOT_BMP);
   flag600    = (ActorVideo *) getActor (ACTOR_600FLAG_BMP);
   flag500    = (ActorVideo *) getActor (ACTOR_FLAG500C_BMP);
   reelfram   = (ActorVideo *) getActor (ACTOR_REELFRAM);
   waitforspin = (ActorVideo *) getActor (ACTOR_WAITFOR_SPIN);
   gmcloud    = (ActorVideo *) getActor (ACTOR_EXPLCLD_BMP);

   // get all actors for game
   sign5c     = (ActorVideo *) getActor (ACTOR_SIGN_FLIC);
   foxexcite  = (ActorVideo *) getActor (ACTOR_FOXEXCITE_FLIC);
   foxreel    = (ActorVideo *) getActor (ACTOR_FOXREEL_FLIC);
   houndreel  = (ActorVideo *) getActor (ACTOR_HOUNDREEL_FLIC);

   houndland  = (ActorVideo *) getActor (ACTOR_HOUNDS_FLIC);
   atgoal     = (ActorVideo *) getActor (ACTOR_LARGE_HND_FLIC);

   foxhop     = (ActorVideoPath *) getActor (ACTOR_FOXHOP_FLIC);
   houndmove  = (ActorVideoPath *) getActor (ACTOR_SMHOUND_FLIC);
   foxsnk     = (ActorVideoPath *) getActor (ACTOR_FOXSNK_FLIC);
   truck      = (ActorVideoPath *) getActor (ACTOR_TRUCK_FLIC);
   winhop     = (ActorVideoPath *) getActor (ACTOR_WINHOP_FLIC);
   drnkin     = (ActorVideoPath *) getActor (ACTOR_DRNKIN_FLIC);
```

Exhibit A
Redacted

```
  cloud     = (ActorVideo *) getActor (ACTOR_CLOUD_FLIC);
  dogsut    = (ActorVideo *) getActor (ACTOR_DOGSUT_FLIC);
  roundp    = (ActorVideo *) getActor (ACTOR_ROUNDP_FLIC);
  foxtst    = (ActorVideo *) getActor (ACTOR_FOXTST_FLIC);
  bigwin    = (ActorVideo *) getActor (ACTOR_BIGWIN_FLIC);
  foxwin    = (ActorVideo *) getActor (ACTOR_FOXWIN_FLIC);

  reel_sound     = (ActorReelSound *) getActor (ACTOR_REEL_SOUND);
  afterhoundspin_sound = (ActorAudio *) getActor (ACTOR_DOGLOOP1_WAV);

  foxexcite_sound = (ActorAudio *) getActor (ACTOR_FXBNCLP1_WAV  );
  theme_sound    = (ActorAudio *) getActor (ACTOR_THEME2B_WAV   );
  foxspin_sound  = (ActorAudio *) getActor (ACTOR_SPIN3_WAV     );
  foxmove_sound  = (ActorAudio *) getActor (ACTOR_FXBNCE1_WAV   );
  houndspin_sound = (ActorAudio *) getActor (ACTOR_SPIN4_WAV     );
  rstop_sound    = (ActorAudio *) getActor (ACTOR_REELSTOP_WAV  );
  houndmove_sound = (ActorAudio *) getActor (ACTOR_DOGBNCE1_WAV  );
  collide_sound  = (ActorAudio *) getActor (ACTOR_FIGHT1_WAV    );
  gowin_sound    = (ActorAudio *) getActor (ACTOR_GOWIN1A_WAV   );
  whine_sound    = (ActorAudio *) getActor (ACTOR_WHINE2_WAV    );
  busted_sound   = (ActorAudio *) getActor (ACTOR_BUSTED1B_WAV  );
  roundup_sound  = (ActorAudio *) getActor (ACTOR_ROUNDUP1C_WAV );
  drinkin_sound  = (ActorAudio *) getActor (ACTOR_DRINKIN1B_WAV );
  foxsneak_sound = (ActorAudio *) getActor (ACTOR_FOXSNK1A_WAV  );
  bigwin_sound   = (ActorAudio *) getActor (ACTOR_BIGWIN1A_WAV  );
  houndclose_sound= (ActorAudio *) getActor (ACTOR_UHOH2_WAV     );
  foxaway_sound  = (ActorAudio *) getActor (ACTOR_BYEBYE1_WAV   );
  foxclose_sound = (ActorAudio *) getActor (ACTOR_FXLAFF1_WAV   );
  tile100_sound  = (ActorAudio *) getActor (ACTOR_100STONE1_WAV );
  tile_sound     = (ActorAudio *) getActor (ACTOR_METER1_WAV    );
  tile0_sound    = (ActorAudio *) getActor (ACTOR_GO1_WAV       );
  flash_sound    = (ActorAudio *) getActor (ACTOR_FLASH1_WAV    );
  gamestart_sound = (ActorAudio *) getActor (ACTOR_GMESTART_WAV  );
  win5c_sound    = (ActorAudio *) getActor (ACTOR_5WIN1_WAV     );

  intro_sound    = (ActorAudio *) getActor (ACTOR_INTRO2A_WAV   );
  introlp_sound  = (ActorAudio *) getActor (ACTOR_INTROLP1_WAV  );
  intro_sound->attach(introlp_sound);

  // getting actor outcome for fox
  ActorArray * array = (ActorArray *)
getActor(ACTOR_FOXREEL_OUTCOME_ARRAY);
  foxreel_outcome_items =  array->getIndexedElementCount ();
  delete [] foxreel_outcome;
  foxreel_outcome = new (ActorVideo * [foxreel_outcome_items]);
```

Exhibit A
Redacted

```
  for (i = 0; i < foxreel_outcome_items; i++)
    foxreel_outcome[i] = (ActorVideo *) array->getActor (i);

  // getting actor outcome for hound
  array = (ActorArray *) getActor(ACTOR_HOUNDREEL_OUTCOME_ARRAY);
  houndreel_outcome_items =  array->getIndexedElementCount ();
  delete [] houndreel_outcome;
  houndreel_outcome = new (ActorVideo * [houndreel_outcome_items]);

  for (i = 0; i < houndreel_outcome_items; i++)
    houndreel_outcome[i] = (ActorVideo *) array->getActor (i);

  // getting hili tiles
  array = (ActorArray *) getActor(ACTOR_HILITILE_ARRAY);
  hilitiles_items =  array->getIndexedElementCount ();
  delete [] hilitiles;
  hilitiles = new (ActorVideo * [hilitiles_items]);

  for (i = 0; i < hilitiles_items; i++)
    hilitiles[i] = (ActorVideo *) array->getActor (i);

  // win meter and lap meter
  array = (ActorArray *) getActor (ACTOR_ARRAY_OF_METERS);
  number_of_meters = array->getIndexedElementCount ();
  delete [] meter_array;
  meter_array = new (ActorGraphicMeter * [number_of_meters]);
  for (i=0; i < number_of_meters; i++)
    meter_array[i] = (ActorGraphicMeter *) array->getActor (i);

  // credit rollup sounds
  array = (ActorArray *) getActor (ACTOR_CREDIT_ROLLUP_SOUND_ARRAY);
  number_of_credit_rollup_sounds = array->getIndexedElementCount ();
  delete [] credit_rollup_sounds;
  credit_rollup_sounds = new (ActorAudio * [number_of_credit_rollup_sounds]);
  for (i = 0; i < number_of_credit_rollup_sounds; i++)
    credit_rollup_sounds[i] = (ActorAudio *) array->getActor (i);

  // credit rollup tag sounds
  array = (ActorArray *) getActor
(ACTOR_CREDIT_ROLLUP_TAG_SOUND_ARRAY);
  number_of_credit_rollup_tag_sounds = array->getIndexedElementCount ();
  delete [] credit_rollup_tag_sounds;
  credit_rollup_tag_sounds = new (ActorAudio *
[number_of_credit_rollup_tag_sounds]);
  for (i = 0; i < number_of_credit_rollup_tag_sounds; i++)
    credit_rollup_tag_sounds[i] = (ActorAudio *) array->getActor (i);
```

Exhibit A
Redacted

```
  // bet one
  array = (ActorArray *) getActor (ACTOR_BET_SOUND_ARRAY);
  number_of_betone_sounds = array->getIndexedElementCount ();
  delete [] betone_sounds;
  betone_sounds = new (ActorAudio * [number_of_betone_sounds]);
  for (i = 0; i < number_of_betone_sounds; i++)
    betone_sounds[i] = (ActorAudio *) array->getActor (i);

  // bet max
  array = (ActorArray *) getActor (ACTOR_BETMAX_SOUND_ARRAY);
  number_of_betmax_sounds = array->getIndexedElementCount ();
  delete [] betmax_sounds;
  betmax_sounds = new (ActorAudio * [number_of_betmax_sounds]);
  for (i = 0; i < number_of_betmax_sounds; i++)
    betmax_sounds[i] = (ActorAudio *) array->getActor (i);

  // loss
  array = (ActorArray *) getActor (ACTOR_LOSS_SOUND_ARRAY);
  number_of_loss_sounds = array->getIndexedElementCount ();
  delete [] loss_sounds;
  loss_sounds = new (ActorAudio * [number_of_loss_sounds]);
  for (i = 0; i < number_of_loss_sounds; i++)
    loss_sounds[i] = (ActorAudio *) array->getActor (i);

  //game play text
  array = (ActorArray *) getActor (ACTOR_GAMEPLAY_TEXT_ARRAY);
  gmtext_items = array->getIndexedElementCount ();
  delete [] gmtext;
  gmtext = new (ActorVideo * [gmtext_items]);
  for (i = 0; i < gmtext_items; i++)
    gmtext[i] = (ActorVideo *) array->getActor (i);

  reelsShow ();
}

//-------------------------------------------------------------------------------------
// Purpose: Create the necessary states for this game.
```

[redacted]

Exhibit A
Redacted

[redacted]

//----------
// Purpose: Process events for this protocol.

[redacted]

Exhibit A
Redacted

Exhibit A
Redacted

[redacted]

//----------

// Purpose:

[redacted]

//----------

Exhibit A
Redacted

// Purpose:
BOOL FoxNHoundGame::processPacket (const Packet &packet)

Exhibit A
Redacted

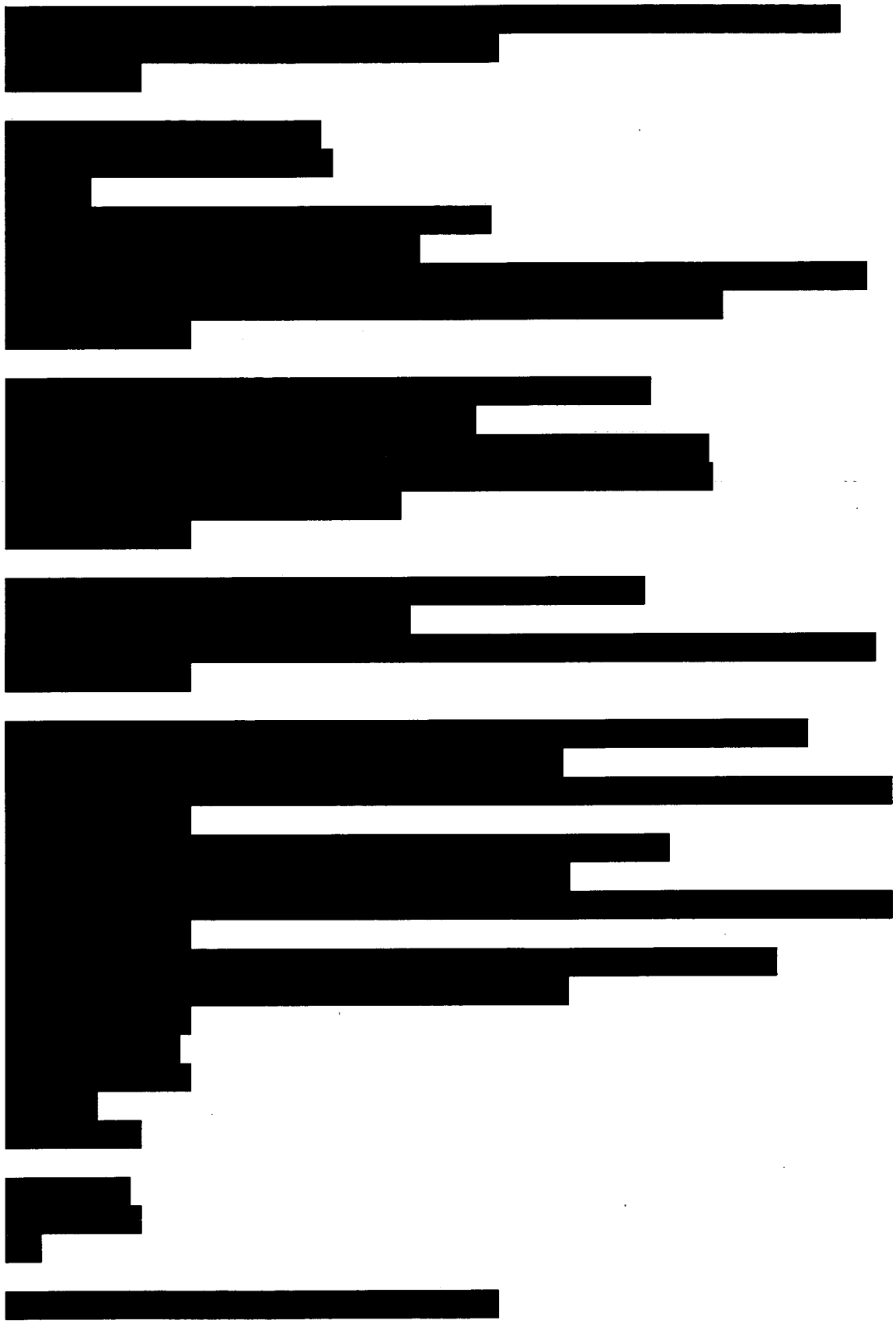


Exhibit A
Redacted

[redacted]

//----------

[redacted]

Exhibit A
Redacted

[redacted]

//----------

[redacted]

//----------

[redacted]

Exhibit A
Redacted

[redacted]

//----------

[redacted]

//----------

[redacted]

Exhibit A
Redacted

[redacted]

```
//--------------------------------------------------------------------------------
// Purpose:
BOOL FoxNHoundGame::isReelWinSoundStopped (void)
{
   if (reel_sound->isReelSoundStopped (reel_sound->REEL_WIN_SOUND))
      return TRUE;

   return FALSE;
```

[redacted]

[redacted]

[redacted]

```
// Purpose: To stop all the credit rollup sounds
```

[redacted]

[redacted]

```
// Purpose: To stop all the credit rollup tag sounds
```

[redacted]

Exhibit A
Redacted

//----------

//----------

```
// Purpose: hide the cloud and text after 2 games play
void FoxNHoundGame::gameplayHideText (void)
{
   if (isSeqDelayDone (10000, startTime))
   {
      show_gmtext = FALSE;
      number_gm_show_text = 0;
      startTime = 0;
      gmcloud->hide ();
      gmtext[gmtext_index]->hide ();
      foxStationaryShow (INIT_FOX_POS, TRUE);
      smallHndStationaryShow (INIT_HOUND_POS);
   }
}
//----------
// Purpose: get fox data from 92X message blocks
void FoxNHoundGame::getFoxPositionData (int pos)
{
   switch (pos)
   {
      case POSPREVIOUS:
         fox_tile_prev_pos = getBlock3PreviousPosition (FOXPOSITION);
         if (fox_tile_prev_pos > MAX_TILES - 1)
            fox_tile_prev_pos = 0;
```

Exhibit A
Redacted

```
        //0 base index received
        foxreel_prev_pos = 1 + getBlock3PreviousPosition (FOXREELPOSITION);
        break;

      case POSCURRENT:
      case POSRESET:
        fox_tile_pos = getBlock3CurrentPosition (FOXPOSITION);
        fox_pos_prop = getBlock3CurrentPositionProperties (FOXPOSITION);
        if (fox_tile_pos > MAX_TILES - 1)
          fox_tile_pos = 0;

        foxreel_pos = 1 + getBlock3CurrentPosition (FOXREELPOSITION);
        fox_tile_reset_pos = getBlock3ResetPosition (FOXPOSITION);
        theme_number = (getBlock3CurrentSubposition (FOXPOSITION) %
MAX_RANDOM_THEME) + 1;
        break;
    }

    lapsmeter = getBlock2LapsCompleted(0);
    winamount = getTotalWinAmount ();      //this amount is always the last win amount.
                                  //Only in showwin will get the current win amount.
}
//--------------------------------------------------------------------------------------
// Purpose: animate fox reel
void FoxNHoundGame::spinFoxReel (void)
{
        foxreel_outcome[0]->hide ();
    foxreel_outcome[foxreel_prev_pos]->hide ();
    foxreel->show ();
    foxreel->loop (TRUE);
    foxreel->playFromFrame (0);
    foxspin_sound->playFromTime (0);
}
//--------------------------------------------------------------------------------------
// Purpose: stop fox reel animation
void FoxNHoundGame::stopFoxReel (void)
{
    foxreel->loop (FALSE);
    foxreel->stop ();
    foxreel->hide ();
}
//--------------------------------------------------------------------------------------
// Purpose: fox reel sound stop, show the outcome of the bonus reel
BOOL FoxNHoundGame::spinFoxReelDone ()
{
        if (foxspin_sound->isStopped ())
```

Exhibit A
Redacted

```
        {
      rstop_sound->playFromTime (0);
      stopFoxReel ();
      foxreel_outcome[foxreel_pos]->show ();

      //if max lap reached, stop fox at 'GO' tile
      if (lapsmeter >= MAX_LAP)
        fox_tile_pos = 0;

      if (fox_pos_prop <= PROPERTY_MAXLAPS_REACH || fox_pos_prop >
PROPERTY_FOXCAUGHTHOUND_HOUNDATGO)
      {
        if (lapsmeter < MAX_LAP)
          hilitiles[fox_tile_pos]->show ();
      }

      checkFoxGetClose ();
      checkFoxGetAway ();

      return TRUE;
    }
    return FALSE;
}
//-----------------------------------------------------------------------------
// Purpose: animate fox excited at introduction state
```

```
//-----------------------------------------------------------------------------
// Purpose:
```

Exhibit A
Redacted

```
//---------------------------------------------------------------------------------------
// Purpose: check fox gets away after hound get too close
void FoxNHoundGame::checkFoxGetAway (void)
{
  if (panic_flag)
  {
    getHoundPositionData (POSRESET);
    int diff = fox_tile_pos - hound_tile_reset_pos;
    if (diff < 0)
      diff = MAX_TILES + (fox_tile_pos - hound_tile_reset_pos);
    if (diff >= FOX_AWAY_DISTANCE)
    {
      fox_getaway = TRUE;
      panic_flag  = FALSE;
    }

  }
}
//---------------------------------------------------------------------------------------
// Purpose: will fox get close to hound
void FoxNHoundGame::checkFoxGetClose (void)
{
  if (fox_pos_prop != PROPERTY_FOXCAUGHTHOUND)
  {
    getHoundPositionData (POSRESET);
    int diff = hound_tile_reset_pos - fox_tile_pos;
    if (diff < 0)
      diff = MAX_TILES + (hound_tile_reset_pos - fox_tile_pos);
    if (diff <= HOUND_IN_PANIC_DISTANCE)
      fox_tooclose = TRUE;
  }
}
//---------------------------------------------------------------------------------------
// Purpose: start collide animation when fox got hound
void FoxNHoundGame::ifFoxGotHound (void)
{
  if ((hound_tile_pos == fox_tile_prev_pos) && (lapsmeter < MAX_LAP))
```

Exhibit A
Redacted

```
   {
      smallHndStationaryHide ();
      foxhop->hide ();
      foxmove_sound->stop ();
      collide_tile_pos    = fox_tile_prev_pos;   //store tile where fox's at rightnow
      fox_tile_prev_pos   = fox_tile_pos;        //to stop fox where the hound is at

      //if (fox_pos_prop != PROPERTY_FOXCAUGHTHOUND_HOUNDATGO)
      if (hound_tile_pos > 0)
      {
         houndmove->setPath (fox_tile_prev_pos);    //do at least the first collide
animation
         houndmove->show ();
         houndmove->play ();
         startCollideSeq ();
      }
   }
}
//-----------------------------------------------------------------------------------------
// Purpose: Move fox to next tile
void FoxNHoundGame::animateFoxHop (void)
{
   if (++fox_tile_prev_pos >= MAX_TILES)
   {
      fox_tile_prev_pos = 0;

     // reach tile 'go' and maxlap, use finishW bitmap
      if ((lapsmeter >= MAX_LAP) && (fox_tile_pos == 0))
      {
         goW->hide ();
         finishW->show ();
      }

     // reach tile 'go' but not max lap, set flag to do 5 credit bonus animation
     // prev tile must be reset first before showLaps called
      if (lapsmeter < MAX_LAP)
      {
         showLaps ();
         fox_at_tile0  = TRUE;      //ref. ifFoxAtTileGo()
      }
   }

   // do animation to next tile
   foxhop->setPath (fox_tile_prev_pos);
   foxhop->show ();
   foxhop->play ();
```

Exhibit A
Redacted

```
   foxmove_sound->playFromTime (0);
}
//-----------------------------------------------------------------------------------------
// Purpose: at tile 'go', setup some flags and timer
void FoxNHoundGame::ifFoxAtTileGo (void)
{
   if (fox_at_tile0)
   {
      if ((fox_pos_prop == PROPERTY_FOXCAUGHTHOUND_HOUNDATGO) &&
(!hound_tile_pos))
         smallHndStationaryHide ();

      fox_at_tile0 = FALSE;
      do_delay_at_tile0 = TRUE;                    //ref. doFoxExciteAtTileGo()
      startTime = system->getTimerMilliseconds ();
      tile0_sound->playFromTime (0);
   }
}
//-----------------------------------------------------------------------------------------
// Purpose: at tile 'go', fox will pause awhile, and bouncing 4 times and get extra 5 credits
[redacted]
         do_delay_at_tile0      = FALSE;
[redacted]
//-----------------------------------------------------------------------------------------
// Purpose: done with excite animation, reset flag
BOOL FoxNHoundGame::doneWithFoxExciteAtTileGo (void)
{
   if (do_foxexcite_at_tile0)
   {
      if (foxexcite->isStopped())
      {
```

Exhibit A
Redacted

```
        do_foxexcite_at_tile0 = FALSE;
        do_credit_at_tile0   = TRUE;      //ref. creditAwardAtTileGo()
        foxExciteHide ();
        sign5c->seekToFrame (0);
        sign5c->show ();
        sign5c->playTillFrame (2);
        gowin_sound->playFromTime (0);
        startTime = system->getTimerMilliseconds ();
      }

      return TRUE;
    }

    return FALSE;
}
//-----------------------------------------------------------------------------------------
// Purpose:  Credit 5 coins, and hopping on
BOOL FoxNHoundGame::creditAwardAtTileGo (void)
{
    if (do_credit_at_tile0)
    {
      if (sign5c->isStopped () && isSeqDelayDone (1500, startTime))
      {
        long extraAmt =  getBlock3ExtraWinAmount(FOXPOSITION);

        meter_array[WIN_METER]->updateGraphicMeter (winamount + extraAmt);
        meter_array[WIN_METER]->show ();

        tile_sound->playFromTime (0);

        do_credit_at_tile0  = FALSE;
        do_delay_credit     = TRUE;
        startTime           = system->getTimerMilliseconds ();
      }

      return TRUE;
    }

    if (do_delay_credit)
    {
      if (sign5c->isStopped () && isSeqDelayDone (1500, startTime))
      {
        sign5c->playFromFrame (2);
        do_delay_credit = FALSE;
        startTime       = 0;
        sign5c->hide ();
```

Exhibit A
Redacted

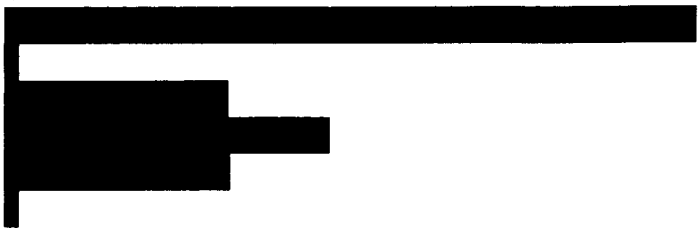
```
      }

      return TRUE;
   }

   return FALSE;
}

//-----------------------------------------------------------------------------------------
// Purpose: animate fox hopping toward the target tiles.  start from tile_prev_pos to
tile_pos
BOOL FoxNHoundGame::foxMoveNTiles (void)
  {
   if (foxhop->isStopped () && foxmove_sound->isStopped())
   {
      ifFoxAtTileGo ();              //set flag to do 5 credit award animation if fox reaches
'go'

      if (doFoxExciteAtTileGo())        //at tile 'go' foxexcite will be animated 4 times
         return FALSE;

      if (doneWithFoxExciteAtTileGo())  //move on to the rest of the steps
         return FALSE;

      if (creditAwardAtTileGo ())
         return FALSE;

      ifFoxGotHound ();

      // prev and cur the same, we've done with animation.
      if (fox_tile_prev_pos == fox_tile_pos)
      {
         hilitiles[fox_tile_pos]->hide ();
         return TRUE;
      }

      animateFoxHop ();

      if (fox_tooclose)
         foxMakeHoundPanic ();
      if (fox_getaway)
         foxJustGetAway ();
   }

   return FALSE;
}
```

Exhibit A
Redacted

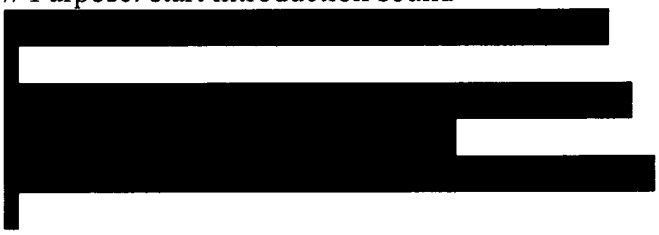
```
//-----------------------------------------------------------------------------
// Purpose: fox go 1 round in the bonus introduction, adding more excitement
BOOL FoxNHoundGame::foxGoCircle (void)
{
  if (foxhop->isStopped ())
  {
    if (fox_go_circle)
    {
      fox_go_circle = FALSE;
      return TRUE;
    }

    if (++fox_gocircle_pos >= MAX_TILES)
    {
      fox_gocircle_pos = 0;
      fox_go_circle = TRUE;
    }

    foxhop->setPath (fox_gocircle_pos);
    foxhop->show ();
    foxhop->play ();
  }

  return FALSE;
}
//-----------------------------------------------------------------------------
// Purpose: animate fox at stationary, the flic will be used at game idle
//      depend on the tile, fox will face either left or right.
//      bDoFlic sets to animate fox toasting when game idles
void FoxNHoundGame::foxStationaryShow (int atTile, BOOL bDoFlic)
{
  if (bDoFlic)
  {
    foxtst->moveGlobal (FOXTST_XCOOR, FOXTST_YCOOR);
    foxtst->playFromFrame (0);
    foxtst->show ();
    foxtst->loop (TRUE);
  }
  else
  {
    foxhop->moveGlobal (foxCoorPos[atTile][0], foxCoorPos[atTile][1]);

    if (atTile < FOXFACERIGHT_TILELOW || atTile >=
FOXFACERIGHT_TILEHIGH)
      foxhop->seekToFrame (FOXFACE_LEFT);
    else
```

Exhibit A
Redacted

```
        foxhop->seekToFrame (FOXFACE_RIGHT);

      foxhop->show ();
   }
}
//-------------------------------------------------------------------------------------------
```

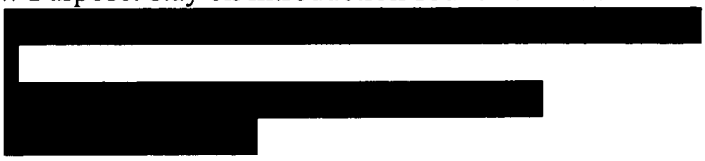


```
//-------------------------------------------------------------------------------------------
// Purpose: start introduction sound
```



```
//-------------------------------------------------------------------------------------------
// Purpose: stop introduction sound
```

```
//-------------------------------------------------------------------------------------------
// Purpose: stay on introduction sound for 4500 ms
```



```
//-------------------------------------------------------------------------------------------
// Purpose: start theme sound
```

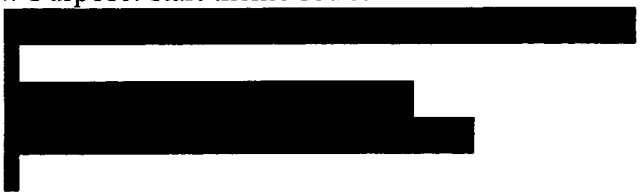


Exhibit A
Redacted

```
//-----------------------------------------------------------------------------
// Purpose: stop theme sound
[redacted]
//-----------------------------------------------------------------------------
// Purpose: start collide sound, this sound will fade at certain time
void FoxNHoundGame::startCollideSound (void)
{
   collide_sound->changeVolume(0);
   collide_sound->loop (TRUE);
   collide_sound->playFromTime(0);
   newAttenuation = 20;
}
//-----------------------------------------------------------------------------
// Purpose: stop theme sound
[redacted]
//-----------------------------------------------------------------------------
// Purpose: play right after hound spin sound
[redacted]
//-----------------------------------------------------------------------------
// Purpose: stop after hound stop moving around the tiles
void FoxNHoundGame::stopAfterHoundSpinSound (void)
{
   afterhoundspin_sound->loop (FALSE);
   afterhoundspin_sound->stop ();
}
//-----------------------------------------------------------------------------
// Purpose: if fox gets to close (within 6 tiles) make sound
void FoxNHoundGame::foxMakeHoundPanic (void)
{
   int diff = hound_tile_reset_pos - fox_tile_prev_pos;

   if (diff < 0)
      diff = MAX_TILES + (hound_tile_reset_pos - fox_tile_prev_pos);
```

Exhibit A
Redacted

```
    if (diff == HOUND_IN_PANIC_DISTANCE)
    {
        foxclose_sound->playFromTime (0);
        fox_tooclose = FALSE;
    }

}
//-----------------------------------------------------------------------------
// Purpose: if hound gets to close (within 3 tiles) make sound
void FoxNHoundGame::houndMakeFoxPanic (void)
{
    int diff = fox_tile_reset_pos - hound_tile_prev_pos;

    if (diff < 0)
        diff = MAX_TILES + (fox_tile_reset_pos - hound_tile_prev_pos);

    if (diff == FOX_IN_PANIC_DISTANCE)
    {
        houndclose_sound->playFromTime (0);
        hound_tooclose = FALSE;
        panic_flag = TRUE;
    }

}
//-----------------------------------------------------------------------------
// Purpose: fox gets away, 6 spaces away
void FoxNHoundGame::foxJustGetAway (void)
{
    int diff = fox_tile_prev_pos - hound_tile_reset_pos;

    if (diff < 0)
        diff = MAX_TILES + (fox_tile_prev_pos - hound_tile_reset_pos);

    if (diff == FOX_AWAY_DISTANCE)
    {
        foxaway_sound->playFromTime (0);
        fox_getaway = FALSE;
    }

}
//-----------------------------------------------------------------------------
// Purpose: Get hound data from 92X message blocks
void FoxNHoundGame::getHoundPositionData (int pos)
{
    switch (pos)
```

Exhibit A
Redacted

```
  {
    case POSPREVIOUS:
      hound_tile_prev_pos = getBlock3PreviousPosition (HOUNDPOSITION);
      if (hound_tile_prev_pos > MAX_TILES - 1)
        hound_tile_prev_pos = 0;

      houndreel_prev_pos = 1 + getBlock3PreviousPosition
(HOUNDREELPOSITION);
      break;

    case POSCURRENT:
    case POSRESET:
      hound_tile_pos = getBlock3CurrentPosition (HOUNDPOSITION);
      hnd_pos_prop = getBlock3CurrentPositionProperties (HOUNDPOSITION);
      if (hound_tile_pos > MAX_TILES - 1)
        hound_tile_pos = 0;

      hound_tile_reset_pos = getBlock3ResetPosition (HOUNDPOSITION);
      houndreel_pos = 1 + getBlock3CurrentPosition (HOUNDREELPOSITION);
      break;
  }
}

//---------------------------------------------------------------------------
// Purpose: animate hound reel spin
void FoxNHoundGame::spinHoundReel (void)
{
```

```
//---------------------------------------------------------------------------
// Purpose:
void FoxNHoundGame::stopHoundReel (void)
{
```

```
//---------------------------------------------------------------------------
// Purpose:
```

Exhibit A
Redacted

```
BOOL FoxNHoundGame::spinHoundReelDone (void)
{
        if (houndspin_sound->isStopped ())
        {
      rstop_sound->playFromTime (0);
      stopHoundReel();
      houndreel_outcome[houndreel_pos]->show ();
      startAfterHoundSpinSound ();
      checkHoundGetClose ();
      return TRUE;
   }
   return FALSE;
}

//-------------------------------------------------------------------------------------
// Purpose: animate hound at stationary
void FoxNHoundGame::smallHndStationaryShow (int atTile)
{
   if (atTile > MAX_TILES)
   {
      system->fatalErrorLog ("FoxNHoundGame::sallHndStationaryShow - Tile position
is greater than MAX_TILES!\n");
      return;
   }

      houndland->moveGlobal (smallhndCoorPos[atTile][0],
smallhndCoorPos[atTile][1]);
   houndland->show ();
   houndland->playFromFrame (0);
   houndland->loop (TRUE);
}
//-------------------------------------------------------------------------------------
// Purpose: hide
```

[redacted]

```
//-------------------------------------------------------------------------------------
// Purpose: will hound get close to fox
void FoxNHoundGame::checkHoundGetClose (void)
{
   if (hnd_pos_prop != PROPERTY_HOUNDCAUGHTFOX)
   {
      getFoxPositionData (POSRESET);
```

Exhibit A
Redacted

```
      int diff = fox_tile_reset_pos - hound_tile_pos;
      if (diff < 0)
         diff = MAX_TILES + (fox_tile_reset_pos - hound_tile_pos);
      if (diff <= FOX_IN_PANIC_DISTANCE)
         hound_tooclose = TRUE;
   }
}
//-----------------------------------------------------------------------------------------
// Purpose: animate hound moving toward the target tile
BOOL FoxNHoundGame::smallHndMoveNTiles (void)
{
   if (houndmove->isStopped () && houndmove_sound->isStopped ())
   {
      //hound caught fox
      if (hound_tile_prev_pos == fox_tile_pos)
      {
         collide_tile_pos = hound_tile_prev_pos;
         hound_tile_prev_pos = hound_tile_pos;
         startCollideSeq ();
      }

      if (hound_tile_prev_pos == hound_tile_pos)
      {
         if (hnd_pos_prop != PROPERTY_HOUNDCAUGHTFOX)
         {
            smallHndStationaryShow (hound_tile_pos);
            houndmove->hide ();
         }

         stopAfterHoundSpinSound ();
         return TRUE;
      }

      if (++hound_tile_prev_pos >= MAX_TILES)
        hound_tile_prev_pos = 0;

      houndmove->setPath (hound_tile_prev_pos);
      houndmove->show ();
      houndmove->play ();
      houndmove_sound->playFromTime (0);

      if (hound_tooclose)
         houndMakeFoxPanic ();
   }

   return FALSE;
```

Exhibit A
Redacted

```
}

//-------------------------------------------------------------------------------------------
// Purpose: start play sound and animate collide sequence
void FoxNHoundGame::startCollideSeq (void)
{
   smallHndStationaryHide ();
   foxmove_sound->stop ();
   foxhop->hide ();
   houndmove->hide ();
   stopAfterHoundSpinSound ();
   stopThemeSound();
   startCollideSound ();
}
//-------------------------------------------------------------------------------------------
// Purpose: hound and fox collide, and moving toward GO tile
BOOL FoxNHoundGame::collideMoveNTiles (void)
{
   if (houndmove->isStopped ())
   {
      houndmove->setPath (collide_tile_pos);
      houndmove->show ();
      houndmove->play ();

      if (collide_tile_pos == 0)
         return TRUE;

      if (collide_tile_pos < MID_TILE)
      {
         if (--collide_tile_pos < 0)
            collide_tile_pos = 0;
      }
      else
      {
         if (++collide_tile_pos > MAX_TILES - 1)
            collide_tile_pos = 0;
      }
   }

   return FALSE;
}
//-------------------------------------------------------------------------------------------
// Purpose: animate until certain delay time, when reach the goal
BOOL FoxNHoundGame::isCollideMoveNTilesDelayDone (void)
{
   if (!isSeqDelayDone (COLLIDE_MAXTIME, startTime))
```

Exhibit A
Redacted

```
    {
      if (houndmove->isStopped ())
      {
        houndmove->setPath (0);
        houndmove->play ();
      }
      return FALSE;
    }

    houndmove->hide ();
    startTime = 0;
    return TRUE;
}

//-----------------------------------------------------------------------------
// Purpose: animate after collide flic reaching 'GO'
void FoxNHoundGame::atGoalLap1Play (void)
{
    houndmove->hide ();
    atgoal->show ();
    atgoal->playNLoops (4);
}

//-----------------------------------------------------------------------------
// Purpose:
BOOL FoxNHoundGame::atGoalLap1Stop (void)
{
    if (atgoal->isStopped ())
    {
      atgoal->hide ();
     return TRUE;
    }

    return FALSE;
}

//-----------------------------------------------------------------------------
// Purpose: animate the cloud
void FoxNHoundGame::playCollideCloud (void)
{
    cloud->setFrameDelay (100);
    cloud->show ();
    cloud->playFromFrame (0);
    whine_sound->playFromTime (0);
}
```

Exhibit A
Redacted

```
//-----------------------------------------------------------------------------
// Purpose: fading collide sound, before the cloud animation is done
void FoxNHoundGame::fadeCollideSound (void)
{
  if (newAttenuation < 0xFFFF)
  {
    DWORD rightch  = newAttenuation;
    DWORD leftch   = (rightch << 16);
    DWORD newvol   = (leftch | rightch);
    collide_sound->changeVolume(newvol);
    newAttenuation += 50;
  }
}
//-----------------------------------------------------------------------------
// Purpose: animate disguise sequence
void FoxNHoundGame::disguiseSeqPlay (void)
{
  playCollideCloud ();

  dogsut->seekToFrame (0);
  dogsut->playOneFrame ();
  dogsut->show ();

  foxsnk->setPath (0);
  foxsnk->seekToFrame (0);
  foxsnk->show ();

  startTime = system->getTimerMilliseconds ();
}

//-----------------------------------------------------------------------------
// Purpose:
BOOL FoxNHoundGame::isDisguiseSeqContinue (void)
{
  if (cloud->isStopped ())
  {
    stopCollideSound ();
    cloud->hide ();
    dogsut->playFromFrame (1);
    foxsneak_sound->playFromTime (0);
    foxsnk->play ();
    return TRUE;
  }
  else
    fadeCollideSound();
```

Exhibit A
Redacted

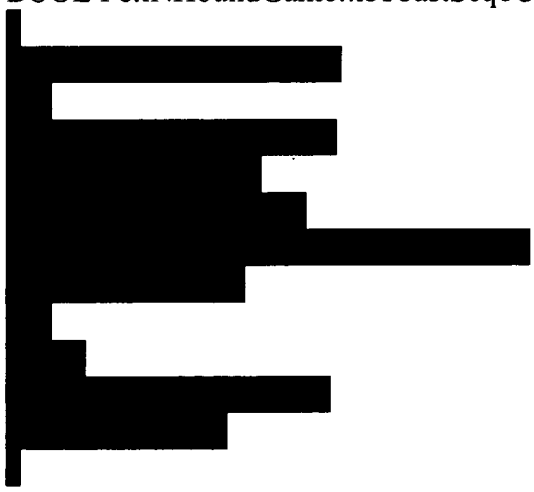
```
    return FALSE;
}
//-----------------------------------------------------------------------------
// Purpose:
BOOL FoxNHoundGame::isDisguiseSeqStop (void)
{
    if (dogsut->isStopped () && foxsnk->isStopped () && foxsneak_sound->isStopped())
    {
        startTime = 0;
        return TRUE;
    }
    return FALSE;
}
//-----------------------------------------------------------------------------
// Purpose: animate round up sequence
void FoxNHoundGame::roundupSeqPlay (void)
{
    playCollideCloud ();

    roundp->seekToFrame (0);
    roundp->playOneFrame ();
    roundp->show ();

    startTime = system->getTimerMilliseconds ();
}

//-----------------------------------------------------------------------------
// Purpose:
BOOL FoxNHoundGame::isRoundupSeqContinue (void)
{
    if (cloud->isStopped ())
    {
        stopCollideSound ();
        cloud->hide ();
        roundp->seekToFrame (0);
        roundp->playNLoops (2);
        roundup_sound->playFromTime (0);
        return TRUE;
    }
    else
        fadeCollideSound();
    return FALSE;
}

//-----------------------------------------------------------------------------
// Purpose:
```

Exhibit A
Redacted

```
BOOL FoxNHoundGame::isRoundupSeqStop (void)
{
   if (roundp->isStopped () && roundup_sound->isStopped())
   {
      startTime = 0;
      return TRUE;
   }
   return FALSE;
}

//-----------------------------------------------------------------------------
// Purpose: animate jail bound sequence
void FoxNHoundGame::jailboundSeqPlay (void)
{
   playCollideCloud ();

   truck->setPath (0);
   truck->seekToFrame (0);
   truck->show ();

   startTime = system->getTimerMilliseconds ();
}

//-----------------------------------------------------------------------------
// Purpose:
BOOL FoxNHoundGame::isJailboundSeqContinue (void)
{
   if (cloud->isStopped ())
   {
      stopCollideSound ();
      cloud->hide ();
      truck->play ();
      busted_sound->playFromTime (0);
      return TRUE;
   }
   else
      fadeCollideSound();
   return FALSE;
}

//-----------------------------------------------------------------------------
// Purpose:
BOOL FoxNHoundGame::isJailboundSeqStop (void)
{
   if (truck->isStopped () && busted_sound->isStopped())
   {
```

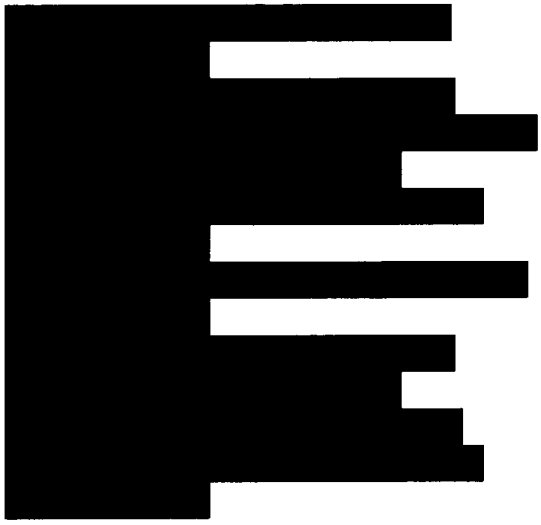
Exhibit A
Redacted

```
        startTime = 0;
        return TRUE;
    }
    return FALSE;
}

//-----------------------------------------------------------------------------
// Purpose: animate toast sequence
void FoxNHoundGame::toastSeqPlay (void)
{
    playCollideCloud ();

    drnkin->setPath (0);
    drnkin->seekToFrame (0);          //show only the first frame
    drnkin->show ();
          drnkin_loop = 0;

    foxtst->moveGlobal (FOXTST_XORG, FOXTST_YORG);
    foxtst->seekToFrame (0);
    foxtst->playOneFrame ();
    foxtst->show ();

    startTime = system->getTimerMilliseconds ();
}

//-----------------------------------------------------------------------------
// Purpose:
BOOL FoxNHoundGame::isToastSeq1Continue (void)
```



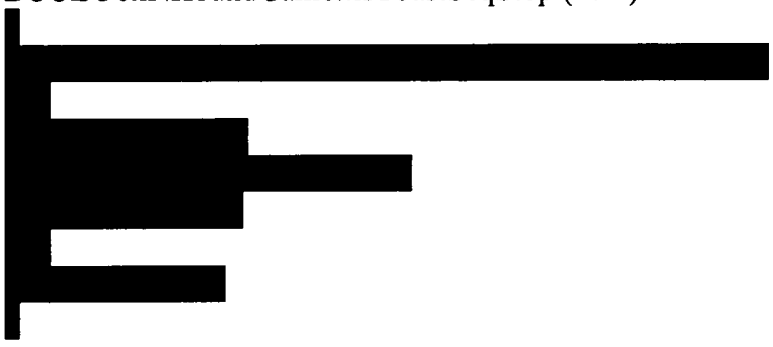
```
//-----------------------------------------------------------------------------
// Purpose:
BOOL FoxNHoundGame::isToastSeq2Continue (void)
```

Exhibit A
Redacted



//---

// Purpose:
BOOL FoxNHoundGame::isToastSeqStop (void)



//---

// Purpose: animate LAST LAP, bigwin. First flash tiles, then fox hops onto podium...

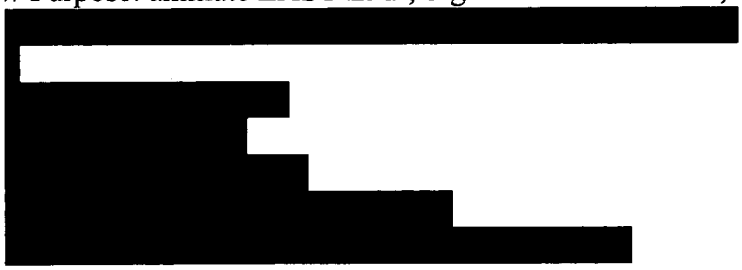


Exhibit A
Redacted

[redacted]

```
//-----------------------------------------------------------------------------
// Purpose: Flashing tiles for 2 seconds with 2 tiles groups: one is even hili tiles,
//          and one is odd.
//          Delay for each group is 333 ms
void FoxNHoundGame::foxwinSeqFlashNextGrp (void)
{
   flashTime = system->getTimerMilliseconds ();
   flash_sound->playFromTime (0);

   if (!bFlashNextGrp)
      foxwinFlashTilesGrp1 (TRUE);
   else
      foxwinFlashTilesGrp2 (TRUE);
}

//-----------------------------------------------------------------------------
// Purpose:
BOOL FoxNHoundGame::isFoxwinSeqFlashTilesDone (void)
{
   if (!isSeqDelayDone (2000, startTime))
   {
      if (isFoxwinFlashTilesGrpDone ())
         foxwinSeqFlashNextGrp ();

      return FALSE;
   }

   reelsHide ();
   foxwinFlashTilesGrp1 (FALSE);
   foxwinFlashTilesGrp2 (FALSE);
   bigwin_sound->playFromTime (0);
   startTime = system->getTimerMilliseconds ();
   return TRUE;
}

//-----------------------------------------------------------------------------
// Purpose:
BOOL FoxNHoundGame::isFoxwinFlashTilesGrpDone (void)
{
   if (!isSeqDelayDone (333, flashTime))
      return FALSE;

   foxwinFlashTilesGrp1 (FALSE);
```

Exhibit A
Redacted

```
    foxwinFlashTilesGrp2 (FALSE);
    bFlashNextGrp = !bFlashNextGrp;

    return TRUE;
}
//-----------------------------------------------------------------------------
// Purpose: Flashing the first group.
```

```
//-----------------------------------------------------------------------------
// Purpose: Flashing the second group.
```

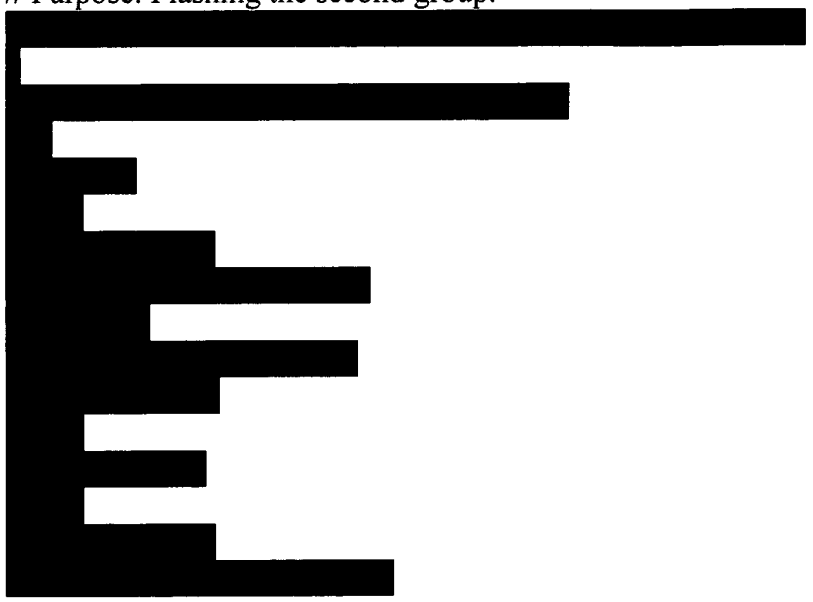


Exhibit A
Redacted

[redacted]

```
}

//-------------------------------------------------------------------------------------
// animate fox hopping on podium
```

[redacted]

```
//-------------------------------------------------------------------------------------
//
BOOL FoxNHoundGame::isFoxwinWinHopDone (void)
```

[redacted]

```
}
//-------------------------------------------------------------------------------------
// animate fox wearing medalist waving...
BOOL FoxNHoundGame::isFoxwinBigWinDone (void)
{
  if (bigwin->isStopped ())
  {
    bigwin->hide ();
    podium->hide ();

    if (lapsmeter >= MAX_LAP)
    {
      // if max lap reach, check for foxcaughthound atgo to display special 500 flag
      if (fox_pos_prop == PROPERTY_FOXCAUGHTHOUND_HOUNDATGO)
      {
```

Exhibit A
Redacted

```
            flag500->show ();
            flag100_bot->show ();
          }
          else
            flag100_top->show ();
      }
      else
      {
        // if fox got hound we also display 500 on top of podium
        if (fox_pos_prop == PROPERTY_FOXCAUGHTHOUND_HOUNDATGO ||
fox_pos_prop == PROPERTY_FOXCAUGHTHOUND)
            flag500->show ();
      }


      foxwin->show ();
      foxwin->playFromFrame (0);
      foxwin->loop (TRUE);
      return TRUE;
    }

    return FALSE;
}
//-----------------------------------------------------------------------------------------
// finishing up and hide everything
BOOL FoxNHoundGame::isFoxwinFinale (void)
{
    if (isSeqDelayDone (7000, startTime))
    {
      if (lapsmeter >= MAX_LAP && fox_pos_prop ==
PROPERTY_FOXCAUGHTHOUND_HOUNDATGO)
      {
        flag500->hide ();
        flag100_top->hide ();
        flag100_bot->hide ();
        flag600->show ();
      }

      foxwin->stop ();
      foxwin->loop (FALSE);
      startTime = 0;
      return TRUE;
    }

    return FALSE;
}
```

Exhibit A
Redacted

```
//-----------------------------------------------------------------------------------------
// Purpose: take care of time delay
BOOL FoxNHoundGame::isSeqDelayDone (DWORD delayTime, DWORD initTime)
{
   stopTime = system->getTimerMilliseconds ();
   if (stopTime - initTime > delayTime)
      return TRUE;

   return FALSE;
}



//-----------------------------------------------------------------------------------------
// hide all flics and bmps used, these bmps might be shown after powerhit or exceptions
void FoxNHoundGame::showWinAnimateHide (void)
{
   atgoal->hide ();
   houndmove->hide ();
   cloud->hide ();
   foxsnk->hide ();
   dogsut->hide ();
   roundp->hide ();
   truck->hide ();
   drnkin->hide ();
   foxtst->hide ();
   winhop->hide ();
   bigwin->hide ();
   foxwin->hide ();
   podium->hide ();
   flag100_bot->hide ();
   flag100_top->hide ();
   flag500->hide ();
   flag600->hide ();
   foxexcite->hide ();
   smallHndStationaryHide ();
   stopFoxReel ();
   stopHoundReel();
   sign5c->hide ();
   fox_at_tile0         = FALSE;
   do_delay_at_tile0    = FALSE;
   do_foxexcite_at_tile0  = FALSE;
   do_credit_at_tile0   = FALSE;
   do_delay_credit      = FALSE;

   finishY->hide ();
```

Exhibit A
Redacted

```
   finishW->hide ();

   for (WORD i = 0; i < MAX_TILES; i++)
      hilitiles[i]->hide ();
}
//-----------------------------------------------------------------------------
// Purpose: show win bitmap
void FoxNHoundGame::showWinAnimateShow (void)
{
   DWORD lastFrame;

   //last lap is a special case, so it does not draw from random number
   if (lapsmeter >= MAX_LAP)
   {
      lastFrame = foxwin->getFrameCount () - 1;
      foxwin->seekToFrame (lastFrame);
      foxwin->show ();

      if (fox_pos_prop == PROPERTY_FOXCAUGHTHOUND_HOUNDATGO)
         flag600->show ();
      else
         flag100_top->show ();

      return;
   }

   //fox got hound is also special case
   if (fox_pos_prop == PROPERTY_FOXCAUGHTHOUND_HOUNDATGO ||
fox_pos_prop == PROPERTY_FOXCAUGHTHOUND )
   {
      lastFrame = foxwin->getFrameCount () - 1;
      foxwin->seekToFrame (lastFrame);
      foxwin->show ();
      flag500->show ();
      return;
   }

   //all the below win is drawn from random number
   switch (theme_number)
   {
      case SHOW_DISGUISE :
          lastFrame = dogsut->getFrameCount () - 1;
         dogsut->seekToFrame (lastFrame);
         dogsut->show ();
         break;
```

Exhibit A
Redacted

```
      case SHOW_ROUNDUP  :
         lastFrame = roundp->getFrameCount () - 1;
         roundp->seekToFrame (lastFrame);
         roundp->show ();
         break;

      case SHOW_TOAST    :
         lastFrame = drnkin->getFrameCount () - 1;
         drnkin->seekToFrame (lastFrame);

         lastFrame = foxtst->getFrameCount () - 1;
         foxtst->seekToFrame (lastFrame);
         foxtst->show ();
         drnkin->show ();
         break;

      case SHOW_JAILBOUND:
         break;
   }
}


//-------------------------------------------------------------------------------------------
// Purpose: hide all bonus bitmaps except background and meters
void FoxNHoundGame::reelsHide (void)
{
   reelfram->hide ();
   arrows->hide ();
   goW->hide ();
   finishY->hide ();
   finishW->hide ();

   for (DWORD i = 0; i < foxreel_outcome_items; i++)
   {
      houndreel_outcome[i]->hide ();
      foxreel_outcome[i]->hide ();
   }
}



//-------------------------------------------------------------------------------------------
// Purpose:
void FoxNHoundGame::reelsShow (void)
{
   arrows->show ();
```

Exhibit A
Redacted

```
   goW->show ();
   reelfram->show ();
   houndreel_outcome[0]->show ();
   foxreel_outcome[0]->show ();
}



//-----------------------------------------------------------------------------------------
// Purpose: show reel and waitforspin button
void FoxNHoundGame::waitForSpinReel (void)
{
   if (!winamount)
   {
      foxExciteShow (0);
      if (houndland->isStopped ())
         smallHndStationaryShow (INIT_HOUND_POS);
   }

         foxreel_outcome[foxreel_pos]->hide ();
         houndreel_outcome[houndreel_pos]->hide ();
         foxreel_outcome[0]->show ();
         houndreel_outcome[0]->show ();
   waitforspin->show ();
}

//-----------------------------------------------------------------------------------------
// Purpose: show at bonus begins
void FoxNHoundGame::setInitialPosition (void)
{
   reelsShow ();

   if (!show_gmtext)
   {
      smallHndStationaryShow (INIT_HOUND_POS);
      foxStationaryShow (INIT_FOX_POS, TRUE);
   }
   else
      gameplayHideText ();

   hound_tooclose  = FALSE;
   fox_tooclose    = FALSE;
   fox_getaway     = FALSE;
   panic_flag      = FALSE;
   winamount = 0;
}
```

Exhibit A
Redacted

```
//-----------------------------------------------------------------------------
// Purpose: everytime fox win credit, play this
void FoxNHoundGame::showMetersSound (void)
{
  if (fox_tile_pos == (MAX_TILES - 1))
    tile100_sound->playFromTime (0);
  else
    tile_sound->playFromTime (0);
}

//-----------------------------------------------------------------------------
// Purpose: everytime fox win credit, show this
void FoxNHoundGame::showMeters (void)
{
  meter_array[WIN_METER]->updateGraphicMeter (winamount);
  meter_array[WIN_METER]->show ();
}

//-----------------------------------------------------------------------------
// Purpose: everytime fox win credit, show this
void FoxNHoundGame::showLaps (void)
{
  BYTE lap;

  //when powerhit don't increment lap if prev pos is greater than cur
  if (fox_tile_prev_pos <= fox_tile_pos)
  {
    if (lapsmeter < MAX_LAP)
      lap = lapsmeter + 1;
    else
      lap = lapsmeter;
  }
  else
    lap = lapsmeter;

  meter_array[LAP_METER]->updateGraphicMeter (lap);
  meter_array[LAP_METER]->show ();

  if ((lapsmeter >= MAX_LAP) && (fox_tile_pos == 0))
  {
    if (fox_tile_prev_pos > fox_tile_pos)
      goW->show ();
    else
    {
      goW->hide ();
```

Exhibit A
Redacted

```
            finishW->show ();
        }
    }
}

//-----------------------------------------------------------------------------
// Purpose: hide credit meters
void FoxNHoundGame::hideMeters (void)
{
        meter_array[WIN_METER]->hide ();
        meter_array[LAP_METER]->hide ();
}
```

This Page is Inserted by IFW Indexing and Scanning Operations and is not part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☐ FADED TEXT OR DRAWING
- ☐ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☐ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☐ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.
As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.